Attorney Docket: SVL920030054US1/2865P

UNITED STATES PATENT APPLICATION

FOR

METHOD AND SYSTEM FOR QUERYING STRUCTURED DOCUMENTS STORED IN THEIR NATIVE FORMAT IN A DATABASE

Inventor(s):

Robbert C. VAN DER LINDEN
Tobias S. MAYR
Brian S. VICKERY

Sawyer Law Group LLP
2465 E. Bayshore Road
Suite 406
Palo Alto, CA 94303

METHOD AND SYSTEM FOR QUERYING STRUCTURED DOCUMENTS STORED IN THEIR NATIVE FORMAT IN A DATABASE

FIELD OF THE INVENTION

The present invention relates generally to computer implemented database systems and, more particularly, to a method and system for querying structured documents stored in their native format in a database system.

BACKGROUND OF THE INVENTION

Structured documents are documents which have nested structures. Documents written in Extensible Markup Language (XML) are structured documents. XML is quickly becoming the standard format for delivering information on the World Wide Web because it allows the user to design a customized markup language for many classes of structure documents. XML supports user-defined tabs for better description of nested document structures and associated semantics, and encourages separation of document contents from browser presentation.

As more and more businesses present and exchange data in XML documents, the challenge is to store, search, and retrieve these documents using the existing relational database systems. A relational database management system (RDBMS) is a database management system which uses relational techniques for storing and retrieving data. Relational databases are organized into tables, which consist of rows and columns of data. A database will typically have many tables and each table will typically have multiple rows and columns. The tables are typically stored "on disk," i.e., on direct access storage devices (DASD), such as magnetic or optical disk drives for semi-permanent storage.

Some relational database systems store an XML document as a BLOB (Binary Large Objects) or map the XML data to rows and columns in one or more relational tables. Both of these approaches, however, have serious disadvantages. First, an XML document that is stored as a BLOB must be read and parsed before it can be queried, thereby making querying costly and time consuming. Second, the mapping process is burdensome and inefficient, especially for large XML documents, because mapping XML data to a relational database can result in a large number of columns with null values (which wastes space) or a large number of tables (which is inefficient). Furthermore, by storing an XML document in a relational database, the nested structure of the document is not preserved. Thus, parent-child(ren) relationships are difficult to reconstruct.

According, there is a need for an improved method and system for querying structured documents stored in their native formats within a database system. The method and system should be integrated (or capable of being integrated) with an existing database system in order to use the existing resources of the database system. The present invention addresses such a need.

SUMMARY OF THE INVENTION

The present invention is directed to a method and system for querying a structured document stored in its native format in a database, wherein the structured document comprises a plurality of nodes that form a hierarchical node tree. The method comprises providing at least one child pointer in each of the plurality of nodes, wherein the at least one child pointer points to a corresponding child node of the plurality of nodes and storing a hint in each of the at least one child pointers. The hint is then utilized to navigate the hierarchical node tree during query evaluation.

Through aspects of the present invention, a structured document is parsed and a plurality of nodes is generated to form a hierarchical node tree representing the structured document. The plurality of nodes is stored in one or more records. Each node that has children includes a plurality of child pointers. Stored in each child pointer is a hint related to the child node to which the child pointer points. In a preferred embodiment, the hint is a portion of the child node's name. By storing the hint in the child pointer, a database management system (DBMS) navigating the node tree during query evaluation follows those pointers that contain a hint that matches a query. Pointers that contain a non-matching hint can be skipped. Accordingly, query processing is more efficient.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of an exemplary computer hardware environment for use with the present invention.

Figure 2 is a block diagram of the XML Storage mechanism according to the preferred embodiment of the present invention.

Figure 3 is a high level flowchart illustrating a process for storing XML documents in their native format in a database according to a preferred embodiment of the present invention.

Figure 4 is a block diagram of an XML Record according to a preferred embodiment of the present invention.

Figure 5 is a block diagram of a page comprising at least one XML Record according to the preferred embodiment of the present invention.

Figure 6 illustrates an exemplary anchor table according to the preferred embodiment of the present invention.

Figures 7A-7C illustrate two pages containing two XML Records according to a preferred embodiment of the present invention.

Figure 8 is a block diagram of a node according to a preferred embodiment of the present invention.

Figure 9 is a flowchart illustrating a process for querying an structured document according to a preferred embodiment of the present invention.

Figure 10 is a flowchart illustrating how a hint is used during query evaluation according to a preferred embodiment of the present invention.

DETAILED DESCRIPTION

The present invention relates generally to computer implemented database systems and, more particularly, to an improved method and system for querying structured documents stored in their native format in a database system. The following description is presented to enable one of ordinary skill in the art to make and use the invention and is provided in the context of a patent application and its requirements. Various modifications to the preferred embodiment and the generic principles and features described herein will be readily apparent to those skilled in the art. For example, the following discussion is presented in the context of a DB2® database environment available from IBM® Corporation. It should be understood that the present invention is not limited to DB2 and may be implemented with other relational database systems or with other native XML database systems. Thus, the present invention is to be accorded the widest scope consistent with the principles and features described herein.

According to a preferred embodiment of the present invention, an XML document is represented by a hierarchical node tree comprising a plurality of nodes. The plurality of nodes are stored in one or more records, which in turn are stored on one or more pages. Each node includes a plurality of child pointers that point to the node's child nodes. Stored in each child pointer is a hint about the child node to which the child pointer points. During query evaluation, the execution of an Xpath or Xquery expression on an XML document translates into navigating the nodes of the node tree. The hint in the child pointer is used to pre-screen child nodes so that those children that may be of interest are accessed.

To describe further the present invention, please refer to Figure 1, which is an exemplary computer environment for use with the present invention. In Figure 1, a typical distributed computer system utilizes a network 103 to connect client computers 102 executing client applications to a server computer 104 executing software and other computer programs, and to connect the server computer 104 to data sources 106. These systems are coupled to one another by various networks, including LANs, WANs, and the Internet. Each client computer 102 and the server computer 104 additionally comprise an operating system and one or more computer programs (not shown).

The server computer 104 uses a data store interface (not shown) for connecting to the data sources 106. The data store interface may be connected to a database management system (DBMS) 105, which supports access to the data store 106. The DBMS 105 can be a relational database management system (RDBMS), such as the DB2® system developed by IBM Corporation, or it also can be a native XML database system. The interface and DBMS 105 may be located at the server computer 104 or may be located on one or more separate machines. The data sources 106 may be geographically distributed.

The DBMS 105 and the instructions derived therefrom are all comprised of instructions which, when read and executed by the server computer 104 cause the server computer 104 to perform the steps necessary to implement and/or use the present invention. While the preferred embodiment of the present invention is implemented in the DB2® product offered by IBM Corporation, those skilled in the art will recognize that the present invention has application to any DBMS, whether or not the DBMS 105 is relational or native. Moreover, those skilled in the art will recognize that the exemplary environment illustrated in Figure 1 is not intended to limit the present invention, and that alternative environments may be used without departing from the scope of the present invention.

According to the preferred embodiment of the present invention, the DBMS 105 includes an XML Storage mechanism 200 that supports the storage of XML documents in their native format on disk. Storing data "on disk" refers to storing data persistently, for example, in the data store 106. Figure 2 is a block diagram of the XML Storage mechanism 200 according to the preferred embodiment of the present invention. The XML Storage mechanism 200 comprises a parser 204 and a node tree generator 206. Each component will be described in further detail in conjunction with Figure 3.

Figure 3 is a flowchart illustrating a method 300 for storing XML documents in their native format in a database according to a preferred embodiment of the present invention. In step 304, the XML Storage mechanism 200 receives or retrieves an XML document 202 for storage, in step 304. The XML document 202 is parsed by the parser 204, (e.g., a standard SAX (Simple API for XML) parser), in step 306. Next, in step 308, the node generator 206 takes the parsed XML data and generates a plurality of nodes that form a hierarchical node tree 208 that represents the XML document 202. In one preferred embodiment, the hierarchical node tree 208 is a DOM (Direct Object Model) tree.

According to the preferred embodiment of the present invention, the node tree 208 preserves the hierarchical structure of the XML document 202 and also preserves the document order, i.e., the order of the nodes. The plurality of nodes forming the node tree 208 is stored in an XML Record 500 in step 310, and each record 500 is, in turn, stored on a page. The XML Record 500 is similar to a standard database record that stores relational data except that the XML Record 500 stores XML data. Storing the plurality of nodes in a record 500 is advantageous because a record 500, like an XML document, is variable in length. Records also can be re-directed, providing a layer of indirection that insulates pointers into a tree, from e.g., within the tree itself, from indices, or from an anchor table (described below), if the record is moved to a different page. Moreover, the infrastructure for fixed page buffer management, recovery, utilities (backup/restore), logging, locking, and replication can be reused.

To explain further the details of the present invention, please refer to Figure 4, which is a block diagram of an XML Record 500 according to a preferred embodiment of the present invention. As is shown, the XML Record 500 comprises a node slot array 506 and a plurality of nodes 508a, 508b, 508 representing at least one XML document 202. Each entry in a node slot 507 points to a node, e.g., 508, in the node tree 208 and provides the following advantages:

- a layer of indirection insulates the pointers into the tree from nodes moving to a different XML Record 500 or page
- a convenient place to find the pointers that might need updating during certain utility functions, e.g., online reorganization, or during in-memory operations on temporary documents where pointers might be replaced with real in-memory pointers
- an inventory of all nodes 508 in an XML Record 500 and of all nodes 508 referenced by nodes 508 in the XML Record 500.

As is shown, each node 508, 508a, 508b comprises an array of child pointers 510. Each child pointer 510 generally points to a node slot 507, which in turn, points to a node, e.g., 508b, corresponding to the child. Thus, for example, in Figure 4, Node B 508b is the child node of Node A 508a. Child pointers 510 can be small because they only need to point to a node slot 507. In certain circumstances a child pointer, e.g., 511, will point to an in-lined character array 512 in the same node, e.g., 508b. The in-lined character array 512 contains information describing the child. In other circumstances, the child pointer 510 itself will comprise information describing the child and its value. The information in the child pointer 510 will be more fully described below.

Figure 5 is a block diagram of a page 502 comprising at least one XML Record 500 according to the preferred embodiment of the present invention. The page 502 includes a record slot array 504 whereby each record slot array entry 505 points to an XML Record 500. According to a preferred embodiment of the present invention, each node 508 in an XML Record 500 on a page 502 is identified by an XID. The XID is a physical identification of the node 508 in an XML document 202. In one preferred embodiment, the

XID comprises a record slot number 505, which points to the XML Record 500, and a node slot number 507, which points to the node 508. In other embodiments, the XID can include a portion of the physical location details of the node 508 where there is an assumed context. Consequently, an XID can be an abstraction of a "node reference," and the node reference itself is implemented differently depending on the circumstances.

A node tree 208 representing an XML document 202 is identified by a root node 508, which is the topmost node 508 in the node tree 208. All other nodes 508 in the node tree 208 are children or descendants of the root node 508. The XID of the root node 508 is referred to as a rootID, and comprises the record slot number 505 pointing to the XML Record 500 and the node slot number 507 pointing to the root node 508.

In another preferred embodiment where the DBMS 105 is a relational database management system, the rootID is stored in an anchor table. Figure 6 illustrates an exemplary anchor table 400 according to the preferred embodiment of the present invention. The anchor table 400 serves as a reference point that "anchors" an XML document 202 within a relational database environment. Each row 406 anchors one XML document 202. As is shown, the anchor table 400 is essentially a relational table comprising an XML Type column 402 and an XML document identifier column 404. The rootID is stored in the XML Type column 402 of the anchor table 400 along with an identifier for the associated XML document 202. Those skilled in the art will recognize that the anchor table 400 is but one way to anchor XML documents 202 within a database system. In a database system other than a relational database system, other formats may be more suitable.

Referring again to Figure 5, the plurality of nodes forming the node tree 208 representing an XML document 202 preferably fits within one XML Record 500 on a page

502. Nevertheless, if the plurality of nodes of the node tree 208 do not fit in one XML Record 500, the plurality of nodes 508 are distributed over a plurality of XML Records 500 on one or more pages 502. To describe how this is accomplished, please refer to Figures 7A-7B, which illustrate two pages 502a and 502b containing two XML Records 500a and 500b according to a preferred embodiment of the present invention. As is shown, the entry in the record slot array 505a points to XML Record 500a. While XML Record 500a is illustrated as containing two nodes 508a, 508b, those skilled in the art readily appreciate that the XML Record 500a is not limited to two nodes and can include one or more than two nodes. The entry in node slot 1 (507a) points to Node A (508a). Node A (508a) has at least two children, one of which is Node B (508b). Child pointer 510a points to node slot 2 (507b), which in turn points to Node B (508b). Each node (e.g., 508b) also points to its parent node (e.g., 508a). Thus, because Node B (508b) is the child node of Node A (508a), Node B (508b) points to node slot 1 (507a), which points to Node A (508a).

Referring now to Figure 7B, Node B (508b) has at least three children. Child pointer 510b points to node slot 4 (507c) whose entry points to a node, e.g., 508c, in a different XML Record 500b on a different page 502b. Thus, the entry in node slot 4 (507c) points to record slot 505b on page 502b, which in turn points to XML Record 500b. The entry also points to node slot number 1 (507c') in XML Record 500b, which then points to Node C (508c).

As is described above, a node slot entry (e.g., 507b) can point to a node (e.g., 508b) that resides within the XML Record 500a, or to a node (e.g., 508c) that resides in a different XML Record 500b. Accordingly, node slot entries (507) are large because they need to be able to point to nodes (508c) in other XML Records 500b in addition to pointing to nodes

(508a, 508b) in the local XML Record 500a. In a preferred embodiment, the entry (e.g., 507a) pointing to a local node (e.g., 508a) is an offset, while the entry (e.g., 507c) pointing to a node (e.g., 508c) in another XML Record 500b is the node's XID. Thus, for example, the entry in node slot 4 (507c) is the XID of Node C, that is, Node C's record slot number and the node slot number.

In the above described example, Nodes A, B and C (508a, 508b, 508c) were distributed over a plurality of XML Records (500a, 500b). In another embodiment, child pointers, e.g., 510a, 510c, of one node, 508a, can be distributed over a plurality of nodes in the same or different XML Records (500a, 500b). This is necessary if the number of child pointers (510a, 510c) for a node (508a) do not fit in the node (508a). Referring now to Figure 7C, one or more child pointers (510c) in a node 508a (referred to as a parent node) point to a continuation node 514, which can reside in the same XML Record (500a) or in a different XML Record 500b. In this instance, the continuation node 514 resides in a different XML Record 500b than that containing the parent node 508a. The continuation node 514 continues the array of child pointers 510d, 510e of the parent node 508a. Thus, the node 508a can have an unlimited number of children (510a, 510c, 510d, 510e) across multiple nodes in the same or different XML Records 500a, 500b on multiple pages 502a, 502b.

For example, referring again to Figure 7C, assume the child pointers (510a, 510c) of Node A (508a) do not fit in Node A (508a). As is shown, child pointer 510c points to node slot number 5 (507d). Because the entry to node slot number 5 (507d) points to a node (e.g., 514) in a second XML Record 500b on a second page 502b, the entry (507d) is preferably the node's XID. Accordingly, the entry (507d) includes a pointer pointing to the record slot

505b on the second page 502b, which in turn points to the second XML Record 500b, as well as a pointer pointing to node slot number 2 (507d') in the second XML Record 500b, which then points to the continuation node 514. Each child pointer 510d, 510e of the continuation node 514 points to a node slot, e.g., child pointer 510d points to node slot 1 (507c'), whose entry points to a node, e.g., Node D (508d). Node D (508d) is a child of Node A (508a) .

The structure and contents of the node 508 will now be described with reference to Figure 8, which is a block diagram of a node 800 according to a preferred embodiment of the present invention. The node 800 is generally divided into two sections, a node header section 802 and a child pointer section 803. The node header section 802 comprises:

- The node type 802a
- The name of the node 802b
- The namespace of this node 802c
- Node slot number of the parent node 802d

The node type (802a) identifies what the node 508 represents in the XML document 202. The node type (802a) is typically an element, attribute, processing instruction or text node, although those skilled in the art recognize that other less common node types exist. The node name (802b) is typically the element or attribute name, or processing instruction target. The namespace of the node (802c) is the namespace of the element or attribute. The node slot number of the parent node (802d) allows the node 508 to identify its parent node for navigation purposes.

The child pointer section 803 comprises one of at least three formats. According to a preferred embodiment of the present invention, there are at least three classes of children:

- The first class is an "ordered" child. Ordered children are children whose order is important pursuant to the XML standard. Ordered children include elements, text, processing instructions and comments type nodes. In addition, proprietary type nodes, e.g., continuation nodes 514, are considered ordered children.
- The second class is an "attribute" child. Attribute type nodes are not ordered.
- The third class is an "internal" child. An internal child is used to store additional information about the parent node, such as a namespace prefix of the node.

In the array of child pointers 510 of the node 508, the order in which children are stored in the node is: "internal" children first, followed by "attribute" children second, and then "ordered" children. This ordering is based on the presumption that the number of internal children is far fewer than the number of attribute children, which in turn is far fewer than the number of ordered children. Thus, child pointers 510 pointing to internal and attribute children will typically be in the main parent node 508, as opposed to a continuation node 514.

Referring again to Figure 8, the format for a child pointer 510 depends on where the child node resides, e.g., in a separate node (508b) or in the parent node (508a). The first format (804) applies when the child pointer, e.g., 510a, points to an ordered, attribute, or internal child in a different node (e.g., 508b). Generally, the node type of such a child will be of an element or attribute type. In this case, the child node (508b) is a node itself. Such a child pointer 510a includes:

- The node slot number of the child node (804a)

- A hint related to the child node (804b)

In a preferred embodiment, a hint about the child node (804b) is stored in the child pointer 510a itself to facilitate navigation during query evaluation. As those skilled in the art are aware, when an XML document 202 is stored in its native format in a database, query evaluation, e.g., execution of an Xpath or Xquery expression, typically involves navigating the nodes 508 of the XML document 202 to find values that satisfy a query. While the cost (resources and time) of navigating between nodes 508 in the same XML Record 500 is relatively inexpensive, the cost of navigating between nodes 508 in different XML Records 500 on different pages 502 is substantial.

The method and system of the present invention addresses this issue. Please refer to Figure 7A and Figure 9, which is a flowchart illustrating a method for querying a structured document according to a preferred embodiment of the present invention. The process begins in step 902 by providing at least one child pointer (510a) in at least one node (508a) of the node tree 208 representing the document 202. Next, a hint (804b) is stored in the at least one child pointer (510a) in step 904. Finally, in step 906, the hint (804b) is utilized during query evaluation to determine whether the DBMS 105 will navigate to the child node (508b) pointed to by the at least one child pointer (510a). By storing the hint about the child node (804b) in the child pointer 510a, the DBMS 105 can decide whether to invest resources to visit the child node (508b) based on the hint. In a preferred embodiment, the hint is a portion of the child node's name because the name is typically descriptive of the node 508.

Figure 10 is a flowchart illustrating how the hint (804b) is used during query evaluation according to a preferred embodiment of the present invention. In step 1002, the DBMS 105 receives a query. Typically, such a query will be an Xpath or Xquery expression although the query could be written in any suitable format known to those skilled in the art. During query evaluation, the DBMS 105 navigates the node tree 208 of an XML document 202 searching for nodes that match the query. In step 1004, the DBMS 105 navigates to a first node, e.g., 508a in Figure 7A, in the node tree 208. There, it checks a first child pointer 510a for a hint (804b) in step 1006. If a hint (804b) exists, the DBMS 105 compares the hint (804b) to the query in step 1008. If a hint (804b) does not exist, e.g, because the child is a processing instruction or a text type node, no comparison is performed.

In the preferred embodiment, the hint (804b) is a portion of the child node's name, e.g., the element or attribute name, because the query most likely comprises tag names and namespaces. Because the hint (804b) is a portion of the node name, the DBMS 105 will seek a partial match of the query. If the hint (804b) at least partially matches the query (in step 1010), there is some likelihood that the child node (508b) is of interest, i.e., the child node (508b) may satisfy the query. Accordingly, the DBMS 105 will follow the child pointer 510a to the child node 508b, and perform a full check, i.e., check the node name and namespace to determine if the child node 508b satisfies the query, via step 1012. If the hint (804b) does not match the query, the child is of no interest, and therefore, the DBMS 105 need not follow the pointer 510a.

Thereafter, in step 1014, if the node 508a has more child pointers (step 1014), the DBMS 105 proceeds to the next child pointer in step 1016, and steps 1008 through 1014 are repeated. When all child pointers in a node have been processed, the DBMS 105 navigates to the next node in step 1018, and steps 1006 through 1018 are repeated until the node tree has been traversed.

By utilizing the hint (804b) stored in the child pointer 510a before following the pointer 510a to the child node 508b, the DBMS 105 avoids visiting nodes that cannot satisfy the query. Accordingly, instead of navigating over the entire node tree 208, the DBMS 105 is able to prune branches of no interest and to navigate to those children that match or partially match the query. Thus, navigation is more efficient and significantly faster.

Referring again to Figure 8 and the different formats for the child pointer 510, the second format (806) applies when the child pointer 510 points to an ordered, attribute or internal child in an in-lined character array 512 in the node (e.g., 508b). Here, the child is part of its parent node. The child pointer 510 fully describes the child, and comprises:

- o The name of the child node, or if it has no name, the type of the node (e.g., text) (806a)
- o The offset and length of the in-lined character array (806b)

The third format (808) is applied when the child pointer 510 itself fully describes the child and its value. In this case, the data in the "pointer" 510 comprises:

- o The name of the child node, or if it has no name, the type of the node (e.g., text) (808a)
- o The child's value (808b)

An improved method and system for querying a structured document stored in its native format in a database is disclosed. Through aspects of the present invention, a structured document is parsed and a plurality of nodes is generated to form a hierarchical node tree representing the structured document. The plurality of nodes is stored in one or more records. Each node that has children includes a plurality of child pointers. Stored in each child pointer is a hint related to the child node to which the child pointer points. In a preferred embodiment, the hint is a portion of the child node's name. By storing the hint in the child pointer, a database management system (DBMS) navigating the node tree during query evaluation follows the pointers that contain a hint that matches a query and can skip over those that contain a non-matching hint. Accordingly, query processing is more efficient.

Although the present invention has been described in accordance with the embodiments shown, one of ordinary skill in the art will readily recognize that there could be variations to the embodiments and those variations would be within the spirit and scope of the present invention. Accordingly, many modifications may be made by one of ordinary skill in the art without departing from the spirit and scope of the appended claims.